CS 3630 : Assignment 4: Simultaneous Localization and Mapping

Deadlines:

- **Check point (Part 1):** Thurs, Feb 19th 11:55pm via T-Square.
- **Final Assignment Due (Part 2)**: Thurs, Feb 26th 11:55pm via T-Square.
- Assignment has to be done in groups of **2**.
- Mention your **group member names** in the submission.

It is absolutely not allowed to share your source code with anyone in the class as well as to use code from the Internet. If you have any questions, ask on Piazza or in class. **Do not give out answers on Piazza!**

Introduction

Simultaneous Localization and Mapping (SLAM) is one of the key robotics problems where one tries to recover the position (or trajectory) of a robot while also constructing a map of the environment at the same time. You'll be working with your partner on this project, using your scribbler to collect data.

1 Record a new dataset (Checkpoint)

A new version of Scribbler2.py has been provided to you. This new class adds the capability to grab grayscale images from the Fluke's camera. The two new functions are **takePicture()** and **savePicture(picture, filename)**

1.1 Build the getAprilTag_mex function

For this assignment we are going to be placing printed tags around the environment called AprilTags (http://april.eecs.umich.edu/wiki/index.php/AprilTags). Each one of these tags provides a unique ID that the library can determine from the pattern, as well as the ability to compute a pose relative to the center of the tag, very useful for trying map out an environment!

We provided a copy of the C library along with a C program that provides the ability to call it from MATLAB. You'll need to compile this program using MATLAB's **mex** function.

In MATLAB, change your directory to the aprilTag source code folder in the matlab directory and execute the following line. Use the line corresponding to Linux/Mac. Feel free to comment on Piazza if you know a better way to compile this which works on all.

Algorithm 1 Compile getAprilTag on Linux

```
mex -v CFLAGS='$CFLAGS -std=gnu99 -D_GNU_SOURCE -fexceptions -fPIC -fno-omit-frame-pointer
-pthread -I' getAprilTag_mex.cpp apriltag.c apriltag_quad_thresh.c g2d.c tag36h11.c tag36artoolkit.c
zarray.c zhash.c zmaxheap.c unionfind.c matd.c image_u8.c pnm.c image_f32.c image_u32.c time_util.c
svd22.c workerpool.c homography.c
```

Algorithm 2 Compile getAprilTag on Mac

```
mex -v getAprilTag_mex.c apriltag.c apriltag_quad_thresh.c g2d.c tag36h11.c tag36artoolkit.c zarray.c zhash.c zmaxheap.c unionfind.c matd.c image_u8.c pnm.c image_f32.c image_u32.c time_util.c svd22.c workerpool.c homography.c
```

If you were successful, you should be able to call the function by passing in a string to the location of an image. Note that it only takes **.pnm** image files, but we will worry about that later.

1.2 Print out some tags and setup your environment.

Print out some of the AprilTags provided in the PDF (`matlab/tags.pdf`) and tape/tack them around a hallway or room. Make sure "Page Scaling" is "None" under the "Page Handling" when printing.

Once you have taped the tags in your environment, replicate it in MATLAB using the **LandmarkMap** class. It takes a matrix with each row defining x,y of each landmark. You can see a simple example at the top of **SLAM.m** in the MATLAB folder. The rows should be in the same indexed order as the index used in the tags. You cannot skip any tag. For example, if you printed first 5 tags from the pdf, the rows in LandmarkMap should correspond to the location of those 5 tags in order.

1.3 Logging the data

You should have plenty of experience in writing a script to move the robot around the environment by now :) Use the functions provided by Scribbler2.py to log motor commands, take pictures, and move around the area.

Some things to keep in mind:

- The camera function **takePicture** can be quite slow (it can take a couple seconds to process and transfer). It's super important the robot remains stationary while the camera is recording, otherwise you will experience an effect called **rolling shutter**, which will corrupt the measurements.
- The **savePicture** function only saves images as **jpg** files, so make sure you use that extension in your filenames.
- The **close** function takes care of converting jpgs to pnm files which can be read by getAprilTag_mex function.
- **logNow** function in this case logs the left and right motor values followed by the filename in case picture is taken. In case no picture is taken it writes **0** as the last value.

1.4 Record two datasets

One important concept in SLAM is that of **loop closure**. When a robot moves through an environment, small measurement errors accumulate in the solution. While not as drastic as the errors we see when just integrating the odometry, over time they can warp and bend the final map.

To correct for these errors, it's important to revisit locations to re-observe earlier landmarks. This effectively snaps the the map into it's correct alignment.

In order to see this in action, we want you to **collect two separate datasets**. First, you should move through the area without revisiting past landmarks. Second, record a dataset where you travel a large distance and then revisit your starting location. Make sure to take plenty of pictures in your dataset. The more tag observations, the more accurate your final solution will be.

1.5 Writeup

1. Take a picture of your real-world environment as well as a screen shot of the simulated replication in MATLAB and include it in your write-up.

2. Make a video of the trajectory taken by the robot (in real environment) when collecting the two datasets. The video should not be more than 5 MB in size.

2 SLAM (Final Submission)

In this part, you'll implement the prediction and update steps of Extended Kalman Filter (**GenericEKF.m**). Before running the SLAM code you have to copy the recorded log file and images from the python directory to matlab directory. You can run the complete SLAM code given the LandmarkMap, log data and pnm file using **SLAM.m**. Make sure to specify the correct log files. Two MATLAB functions have been provided to you. The first is **getRangeAndBearing** which returns the tagID and a bearing and range measurement to the center of the tag given the filename of the image. It is called from **reading()** function of **GenericRangeBearingSensor.m**. Verify if range and bearing measurements given by **getRangeAndBearing** function are correct by placing an AprilTag at a known range and bearing. Second, a forward kinematics function has been provided to help you compute the relative pose. Change the scaling factor to what you experimented in Assignment 3. It can be changed through line 43 in **DeterminsticPath.m**.

Deliverable

1. Implement the prediction and update steps in **update()** function. Specifically implement lines 715, 718, 721 corresponding to **prediction** step and lines 763, 764, 809, 812, 815, 823 corresponding to **update** step.

2. In the write-up, provide visualizations of the mean and covariance estimates of the trajectory and landmarks for both of your datasets.

3. What difference do you see in the estimates with and without loop closure? Provide some reason behind the difference (if any). Include screenshots and justify your argument.

4. Attach all of the code you have written plus your datasets, we should be able to reproduce your results when grading.